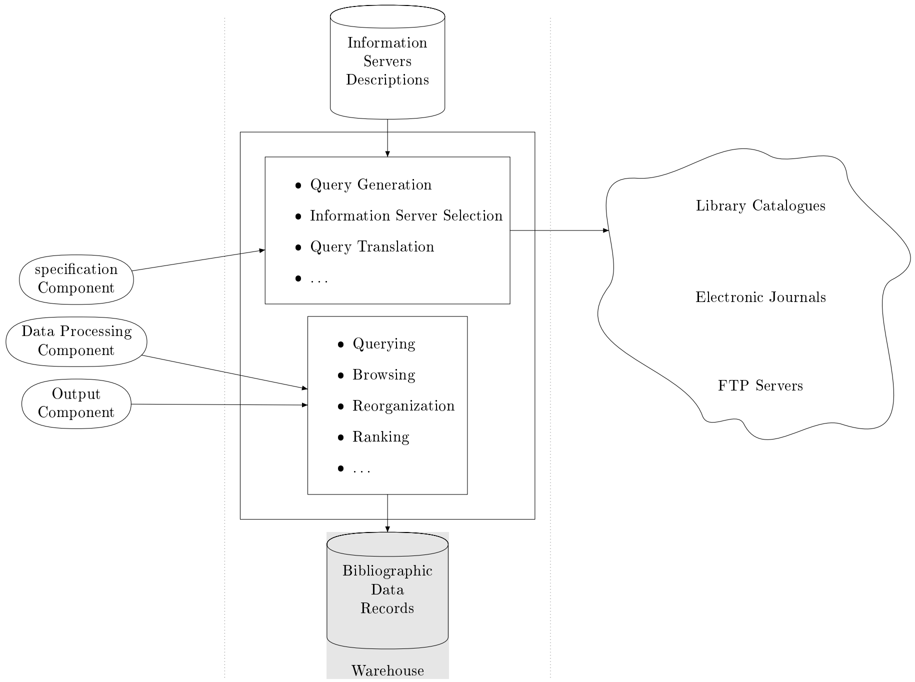
functional METAPOST
Tutorial

Joachim Korittky

May 27, 2002

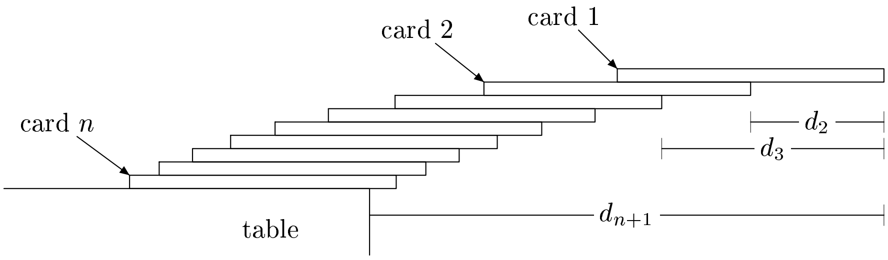
Contents

```
dbs                = draw [arrow ("Information" ◁ C) ("Query" ◁ C),
                          arrow ("Spec" ◁ C) ("Query" ◁ C),
                          arrow ("Proc" ◁ C) ("Querying" ◁ C),
                          arrow ("Output" ◁ C) ("Querying" ◁ C),
                          arrow ("Querying" ◁ C) ("Bib" ◁ C),
                          arrow ("Query" ◁ C) (xy (ref ("Servers" ◁ C)) (ref ("Query" ◁ C)))
                                # setEndCut "Servers",
                          line (xy (ref ("Center" ◁ W) − vec (10,0)) (ref ("Information" ◁ N)))
                               (xy (ref ("Center" ◁ W) − vec (10,0)) (ref ("Warehouse" ◁ S)))
                                # setPattern dotted,
                          line (xy (ref ("Center" ◁ E) + vec (10,0)) (ref ("Information" ◁ N)))
                               (xy (ref ("Center" ◁ E) + vec (10,0)) (ref ("Warehouse" ◁ S)))
                                # setPattern dotted
                          ]
                     (cdrop (0,0) frontEnd & cdrop (200,0) backEnd
                   &  cdrop (440,30) servers)
      where
      frontEnd     = (oval (texCol "specification\\\\Component") # setName "Spec")
                   □ (oval (texCol "Data␣Processing\\\\Component") # setName "Proc")
                   □ (oval (texCol "Output\\\\Component") # setName "Output")
      backEnd      =  setTrueBoundingBox (drum (texCol "Information\\\\Servers\\\\Descriptions
                           # setName "Information")
                   □ (box ((box ("\\parbox{150pt}{\\begin{itemize}\\item␣Query␣Generation"+
                        "\\item␣Information␣Server␣Selection\\item␣Query␣Translation"++
                        "\\item$\\ldots$\\end{itemize}}") # setName "Query")
                   □  (box ("\\parbox{95pt}{\\begin{itemize}\\item␣Querying\\item␣Browsing
                      "\\item␣Reorganization\\item␣Ranking\\item$\\ldots$\\end{itemize}}"
                      # setName "Querying")) # setName "Center" # setDX 16 # setDY 16)
                   □ (setTrueBoundingBox (drum (texCol "Bibliographic\\\\Data\\\\Records")
                      # setName "Bib")
                   □ "Warehouse" # setBGColor 0.9 # setName "Warehouse")
      servers      = fuzzy 1 2 (tex "Library␣Catalogues" □ vspace 30
                          □tex "Electronic␣Journals" □ vspace 30
                          □tex "FTP␣Servers") # setName "Servers"
      texCol c     = tex ("\\begin{tabular}{c}" ++ c ++ "\\end{tabular}")
```

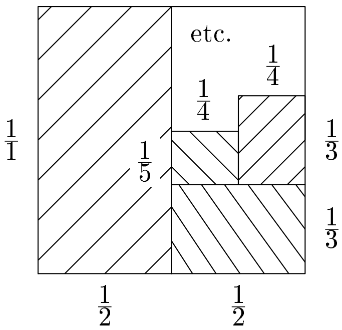


0.1 Introduction

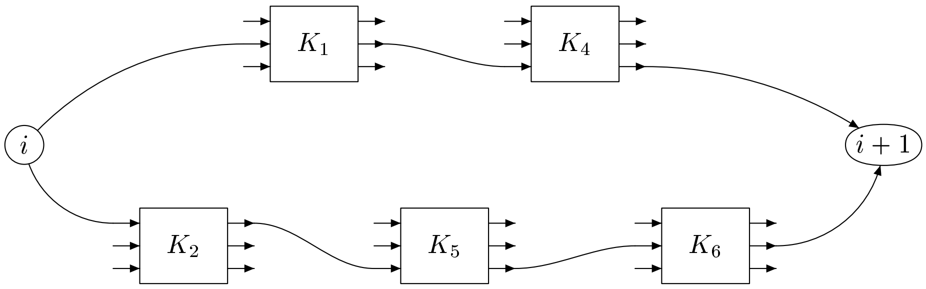
```
conc259              = draw [arrow ("1" ◁ C) (dk !! 0, 0),
                              arrow ("2" ◁ C) (dk !! 1, −5),
                              arrow ("n" ◁ C) (dk !! 8, −40)]
                        (cards
                     & cdraw (darrow (100 + dk !! 1, −20) (0, −20) "d_2")
                     & cdraw (darrow (100 + dk !! 2, −30) (0, −30) "d_3")
                     & cdraw (darrow (100 − 100 ∗ d 11, −55) (0, −55) "d_{n+1}")
                     & cdraw ((−330, −45) -- (100 − 100 ∗ d 11, −45)
                                   -- (100 − 100 ∗ d 11, −70))
                     & cdrop (−230, −60) "table"
                     & cdrop (−120, 20) (tex "card␣$1$" # setName "1")
                     & cdrop (−175, 15) (tex "card␣$2$" # setName "2")
                     & cdrop (−310, −20) (tex "card␣$n$" # setName "n"))
    where
    cards            = foldl (&) relax [cdraw ((x, y) -- (x + 100, y)
                              --(x + 100, y − 5) -- (x, y − 5) -- cycle)
                                   | (x, y) ← zip dk [0, −5 ..]]
    dk               = [−100 ∗ d i | i ← [2 .. 10]]
    d 1              = 0
    d n              = (n − 1 + sum [d i | i ← [1 .. (n − 1)]]) / (n − 1)
    darrow a b s     = a -- b # setArrowHead (arrowHeadSize 4 180)
                              # setStartArrowHead (arrowHeadSize 4 180)
                              # setLabel 0.5 C (math s # setBGColor white)
```



```
conc66                  =   box (0,0) (50,100) (λi → (0,200 − i ∗ 1.5))
                                           (λi → (i ∗ 1.5,200))
                        &   box (50,0) (100,33.3) (λi → (0,i)) (λi → (i,−50))
                        &   box (75,33.3) (100,66.6) (λi → (0,100 − i)) (λi → (i,100))
                        &   box (50,33.3) (75,53.3) (λi → (0,i)) (λi → (i,0))
                        &   cdraw (vec (50,100) -- vec (100,100) -- vec (100,50))
                        &   frac (−10,50) 1
                        &   frac (25,−12) 2
                        &   frac (75,−12) 2
                        &   frac (110,17) 3
                        &   frac (110,50) 3
                        &   frac (62,65) 4
                        &   frac (88,78) 4
                        &   frac (40,42) 5
                        &   cdrop (65,90) "etc."
     where
     frac p n           =   cdrop p (math ("\\frac{\\textstyle␣1}{\\textstyle␣"
                                  ++ show n ++ "}") # setBGColor white)
     box (ax,ay) (bx,by) fa fb =  cdrop (0.5 ∗ (ax + bx),0.5 ∗ (ay + by))
                                (cdraws [fa i -- fb i|i ← [−50,−40 . . 200]]
                                &cclip p
                                &cdraw p)
         where
         p              =   vec (ax,ay) -- vec (bx,ay) -- vec (bx,by)
                            --vec (ax,by) -- cycle
```

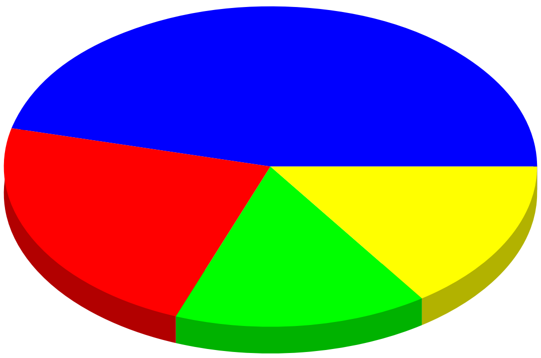


```
hamilton         = arrows (matrix [[empty, empty, k 1, empty, k 4],
                                   [i ▭ hspace 20, empty, empty, empty,
                                    empty, empty, hspace 20 ▭ ip1],
                                   [empty, k 2, empty, k 5, empty,
                                    k 6 # setName "6"]])
   where
   i             = circle (math "i") # setName "i"
   ip1           = oval (math "i+1") # setName "ip1"
   k s           = boxArrows s (box (math ("K_" ++ show s))
                   #setDX 8
                   #setDY 8)
   arrows        = draw [("i" ◁ C) .. (2 ◁ 1 ◁ C) # setEndAngle 0,
                         ("i" ◁ C) .. (1 ◁ 2 ◁ C) # setEndAngle 0,
                         curve (1 ◁ 5 ◁ C) 0 0 (4 ◁ 3 ◁ C),
                         curve (2 ◁ 4 ◁ C) 0 0 (5 ◁ 3 ◁ C),
                         curve (5 ◁ 6 ◁ C) 0 0 (6 ◁ 2 ◁ C),
                         arrow (4 ◁ 6 ◁ C) ("ip1" ◁ C) # setStartAngle 0,
                         arrow (6 ◁ 5 ◁ C) ("ip1" ◁ C) # setStartAngle 0]
   curve x a1 a2 y = x .. y # setStartAngle a1 # setEndAngle a2
   boxArrows     :: Int → Frame → Picture
   boxArrows s   = setName s ∘ draw [out 4 0.2, out 5 0.5, out 6 0.8,
                                     to 1 0.2, to 2 0.5, to 3 0.8]
      where
      out m n    = arrow (med n (ref NE) (ref SE))
                         (med n (ref NE) (ref SE) + vec (10, 0))
                   #setLabel 1 C (empty # setName (m :: Int))
      to m n     = arrow (med n (ref NW) (ref SW) − vec (10, 0))
                         (med n (ref NW) (ref SW))
                   #setLabel 0 C (empty # setName (m :: Int))
```



```
torte = cake [(30, blue), (15, red), (10, green), (10, yellow)]
    where
    cake as = drawBack (toAngles as)
            & drawCake (toAngles as)
    toAngles as = ad (map (λ   (   a, c) → (a / (sum (map fst as) / 360), c)) as)
    ad [] = [(0, black)]
    ad ((a, c) : as) = (a + fst (head (ad as)), c) : ad as
    drawBack [] = relax
    drawBack [_] = relax
    drawBack ((a, c) : (b, c2) : as)
                  = cfill   (   toArea (carc' (vec (0, −10)) 100 60 a b -- carc' (vec (0, 0)) 100 40 b a -- cycle)
                            #   setColor (c − grey 0.3)) & drawBack ((b, c2) : as)
    drawCake [] = relax
    drawCake [_] = relax
    drawCake ((a, c) : (b, c2) : as)
                  = cfill (toArea (vec (0, 0) -- carc' (vec (0, 0)) 100 60 a b -- cycle)
                            #   setColor c) & drawCake ((b, c2) : as)

carc' :: Point → Numeric → Numeric → Numeric → Numeric → Path
carc' p rx ry w1 w2          =  (p + vec (rx ∗ cos a1, −ry ∗ sin a1)
                             ..  p + vec (rx ∗ cos a2, −ry ∗ sin a2)
                             ..  p + vec (rx ∗ cos a2b, −ry ∗ sin a2b)
                             ..  p + vec (rx ∗ cos a3, −ry ∗ sin a3)
                             ..  p + vec (rx ∗ cos a4b, −ry ∗ sin a4b)
                             ..  p + vec (rx ∗ cos a4, −ry ∗ sin a4)
                             ..  p + vec (rx ∗ cos a5, −ry ∗ sin a5))
        where
        a1                   =  w1
        a2                   =  w1 + 0.05 ∗ (w2 − w1)
        a2b                  =  w1 + 0.25 ∗ (w2 − w1)
        a3                   =  w1 + 0.5 ∗ (w2 − w1)
        a4b                  =  w1 + 0.75 ∗ (w2 − w1)
        a4                   =  w1 + 0.95 ∗ (w2 − w1)
        a5                   =  w2
```



0.2 Atomic pictures

The basic data type of the *functional* METAPOST language is *Picture* or shorter, *Pic*. Expressions of this type describe pictures, which can be copiled to METAPOST–code or combined to new more complex pictures.
Atomic pictures are those not generated by combination or modification of other pictures.
Two such atomic expressions are the embedding of arbitrary LaTeX expressions and the generation of an empty rectangle of given size:

$$\begin{aligned} &tex && :: \; String \to Picture \\ &space && :: \; Numeric \to Numeric \to Picture \end{aligned}$$

Every picture in *functional* METAPOST has a surrounding bounding box.[1] Bounding boxes allow the combination of pictures without accidental overlaps.
The two atomar functions *tex* and *space* can be extended easily. The following function allows the use of the TeX math mode

$$\begin{aligned} &math && :: \; String \to Picture \\ &math\ p && = \; tex\ (\texttt{"\$"} +\!\!+\ p +\!\!+ \texttt{"\$"}) \end{aligned}$$

Also defined are functions for horizontal or vertical space (with names borrowed from LaTeX)

$$\begin{aligned} &hspace, vspace && :: \; Numeric \to Picture \\ &hspace\ n && = \; space\ n\ 0 \\ &vspace\ n && = \; space\ 0\ n \end{aligned}$$

This will be useful, too:

$$\begin{aligned} &empty && :: \; Picture \\ &empty && = \; space\ 0\ 0 \end{aligned}$$

0.3 Colors

Every visible object in *functional* METAPOST has a color. The default is *black*. Colors are described by their RGB values, i.e. by additive mixing of the basic colors red, green and blue:

$$color \; :: \; Double \to Double \to Double \to Color$$

Some colors are predefined:

$$\begin{aligned} &white && = \; color\ 1\ 1\ 1 \\ &black && = \; color\ 0\ 0\ 0 \\ &red && = \; color\ 1\ 0\ 0 \\ &green && = \; color\ 0\ 1\ 0 \\ &blue && = \; color\ 0\ 0\ 1 \\ &yellow && = \; color\ 1\ 1\ 0 \\ &cyan && = \; color\ 0\ 1\ 1 \\ &magenta && = \; color\ 1\ 0\ 1 \\ &grey\ n && = \; color\ n\ n\ n \end{aligned}$$

The type *Color* is an instance of *Num* and *Fractional*. We can add, subtract and multiply colors. This happens componentwise. Therefore, $red + green$ gives *yellow* and $cyan - blue$ denotes the color *green*. The function *fromRational* is implemented, too: $0.5 * magenta$ is interpreted as $grey\ 0.5 * magenta$, giving $color\ 0.5\ 0\ 0.5$.
The color of a picture is changed by

[1] A bounding box in *functional* METAPOST is not necessarily rectangular. Triangular and circular bounding boxes are also possible.

$$setColor \quad :: \quad Color \to Picture \to Picture$$

Properties of pictures, like *Color*, are called attributes and modified by attribute functions. One such attribute function, setting the background color, is

$$setBGColor \quad :: \quad Color \to Picture \to Picture$$

Attribute functions are more readable using a postfix notation and the # operator.

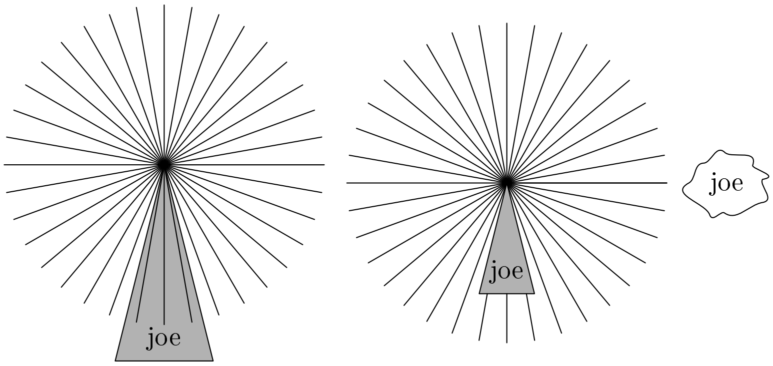
$$\begin{array}{lll} (\#) & :: & a \to (a \to b) \to b \\ a \, \# f & = & f \; a \end{array}$$

$$\begin{array}{lll} color1 & = & tex \; \texttt{"colors"} \\ & & \#setColor \; green \\ & & \#setBGColor \; 0.2 \end{array}$$

Figure 0.3.1: Color attribute functions

0.4 Frames

$$box, circle, oval \quad :: \quad Picture \to Picture$$



$$\begin{array}{lll} newshapes & = & shape \; (setDX \; 10 \circ triangle) \\ & & \quad □□shape \; (triangle) \\ & & \qquad □□cloud \; 345 \; 67 \; (tex \; \texttt{"joe"}) \end{array}$$

$$\begin{array}{lll} shape & :: & (IsPicture \; a, HasBGColor \; a) \\ & \Rightarrow & (Picture \to a) \to Picture \\ shape \; f & = & setTrueBoundingBox \\ & & \quad (draw \; [ref \; C \; \text{--} \; ref \; C + 60 * dir \; a \\ & & \qquad | a \leftarrow [0, 10 \, .. \, 360]] \\ & & \qquad (f \; (tex \; \texttt{"joe"}) \, \# \, setBGColor \; 0.7)) \end{array}$$

0.5 Combination of pictures

Up to now, we created atomic pictures and added frames. Let us now combine two pictures to a new one. The most common combinators set two pictures side by side or on top of the other.

$$
\begin{array}{lcl}
(\boxbar) & :: & \mathit{Picture} \to \mathit{Picture} \to \mathit{Picture} \\
(\boxminus) & :: & \mathit{Picture} \to \mathit{Picture} \to \mathit{Picture}
\end{array}
$$

Sometimes we want a small distance between the pictures:

$$
\begin{array}{lcl}
(\square\square) & :: & \mathit{Picture} \to \mathit{Picture} \to \mathit{Picture} \\
a \;\square\square\; b & = & a \boxbar \mathit{hspace}\ 8 \boxbar b \\[1ex]
(\genfrac{}{}{0pt}{}{\square}{\square}) & :: & \mathit{Picture} \to \mathit{Picture} \to \mathit{Picture} \\
a \genfrac{}{}{0pt}{}{\square}{\square} b & = & a \boxminus \mathit{vspace}\ 8 \boxminus b
\end{array}
$$

Now it is time to discuss units. METAPOST uses as basic unit PostScript points which correspond to $1/72$ inch. We use them in *functional* METAPOST, too. *hspace* 8 defines a horizontal distance of $1/9$ inch or approximately 2.82 mm. Some predefined constants allow the use of other units

$$
\begin{array}{lcl}
mm, pt, bp, cm, pc, cc, inch & :: & \mathit{Numeric} \\
mm & = & 2.84528 \\
pt & = & 1 \\
bp & = & 1.00375 \\
cm & = & 28.45276 \\
pc & = & 12 \\
cc & = & 12.84010 \\
inch & = & 72.27
\end{array}
$$

hspace $(2 * cm)$ generates a distance of two centimeters.

$$
\begin{array}{lcl}
\mathit{bsp30} & = & \mathit{tree1} \\
 & \square\square & \mathit{hspace}\ 10 \\
 & \square\square & (\texttt{"\$\textbackslash\textbackslash Longrightarrow\$"} \genfrac{}{}{0pt}{}{\square}{\square} \texttt{"right\textvisiblespace rotation"}) \\
 & \square\square & \mathit{hspace}\ 10 \\
 & \square\square & \mathit{tree2}
\end{array}
$$

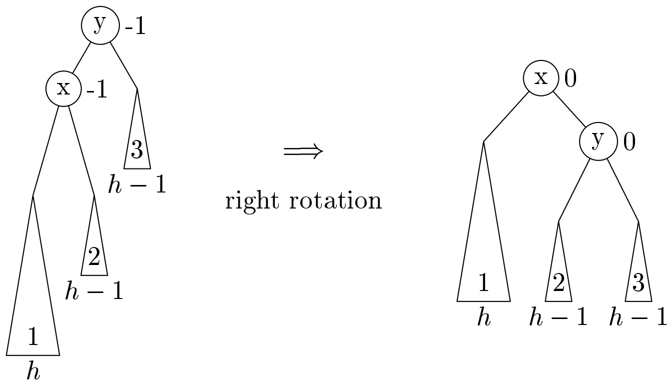


Figure 0.5.1: Placements

```
bsp40        = box2 (tex "set␣of␣all␣languages"
             □    box2 (tex "rec.␣enum.␣languages␣or␣typ␣0"
             □    box2 (tex "decideable␣languages"
             □    box2 (tex "contextsensitive␣or␣typ␣1"
             □    box2 (tex "contextfree␣or␣typ␣2"
             □    box2 "regular␣or␣typ␣3--languages")))))
      where
      box2 a = rbox 20 a
             # setDX 8
             # setDY 8
```

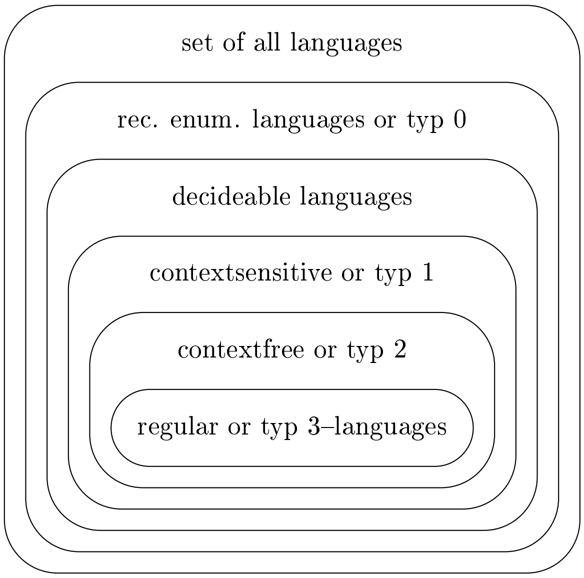


Figure 0.5.2: Combinations of frames and placements

The combinators have generalizations for more than two pictures:

row	::	$[Picture] \to Picture$
column	::	$[Picture] \to Picture$

Fig. 0.5.3 shows a variant with additional space between the pictures

rowSepBy	::	$Numeric \to [Picture] \to Picture$
columnSepBy	::	$Numeric \to [Picture] \to Picture$

bsp60 = *box* (*columnSepBy* 10
 [*circle* "R"
 # *setDX* 10
 # *setBGColor red*,
 circle "O"
 # *setDX* 10
 # *setBGColor yellow*,
 circle "G"
 # *setDX* 10
 # *setBGColor green*])
 #*setBGColor black*
 #*setDX* 10
 #*setDY* 10

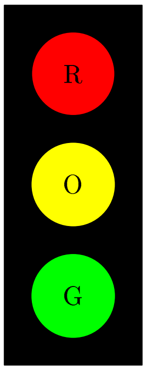


Figure 0.5.3: A traffic light

Sometimes we need a two dimensional alignment of pictures, as in Fig. 0.5.4.

$$matrix \quad :: \quad [[Picture]] \to Picture$$

The function *matrix* generates columns and rows. Every column has the width of the picture with the largest width in it. Analogously, every row has the height of the highest picture in it. Then the pictures are centered in the so-defined rectangular cells.
There is also a variant with additional space:

$$matrixSepBy \quad :: \quad Numeric \to Numeric \to [[Picture]] \to Picture$$

matrix1 = *matrix* [[*tex* "a", *tex* "b"],
 [*empty*, *tex* "c", *tex* "d"]]

a b
c d

Figure 0.5.4: Alignment with the *matrix* function

0.6 Paths

type Point,

$$vec \quad :: \quad (Numeric, Numeric) \to Point$$

$$med \quad :: \quad Numeric \to Point \to Point \to Point$$

type Numeric

$$xpart \quad :: \quad Point \to Numeric$$

$$ypart \quad :: \quad Point \to Numeric$$

$$angle :: Point \to Numeric$$

$$dist :: Point \to Point \to Numeric$$

$$med :: Numeric \to Numeric \to Numeric \to Numeric$$

composite functions:

$$\begin{array}{lcl} dir & :: & Numeric \to Point \\ dir\ a & = & vec\ (cos\ a, sin\ a) \end{array}$$

$$\begin{array}{lcl} xy & :: & Point \to Point \to Point \\ xy\ p_1\ p_2 & = & vec\ (xpart\ p_1, ypart\ p_2) \end{array}$$

$$\begin{array}{lcl} xdist & :: & Point \to Point \to Numeric \\ xdist\ p_1\ p_2 & = & xpart\ p_1 - xpart\ p_2 \end{array}$$

$$\begin{array}{lcl} ydist & :: & Point \to Point \to Numeric \\ ydist\ p_1\ p_2 & = & ypart\ p_1 - ypart\ p_2 \end{array}$$

Syntax for paths:

$$(\text{--}) :: (IsPath\ a, IsPath\ b) \Rightarrow a \to b \to Path$$

$$(..) :: (IsPath\ a, IsPath\ b) \Rightarrow a \to b \to Path$$

$$(\text{---}) :: (IsPath\ a, IsPath\ b) \Rightarrow a \to b \to Path$$

$$(...) :: (IsPath\ a, IsPath\ b) \Rightarrow a \to b \to Path$$

$$(\&) :: (IsPath\ a, IsPath\ b) \Rightarrow a \to b \to Path$$

$$cycle :: Path$$

$$\begin{array}{lcl} path1 & = & vec\ (20,20)\ \text{--}\ vec\ (0,0)\ \text{--}\ vec\ (0,30) \\ & \text{--} & vec\ (30,0)\ \text{--}\ vec\ (0,0) \end{array}$$

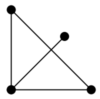


Figure 0.6.1: A simple path of line segments

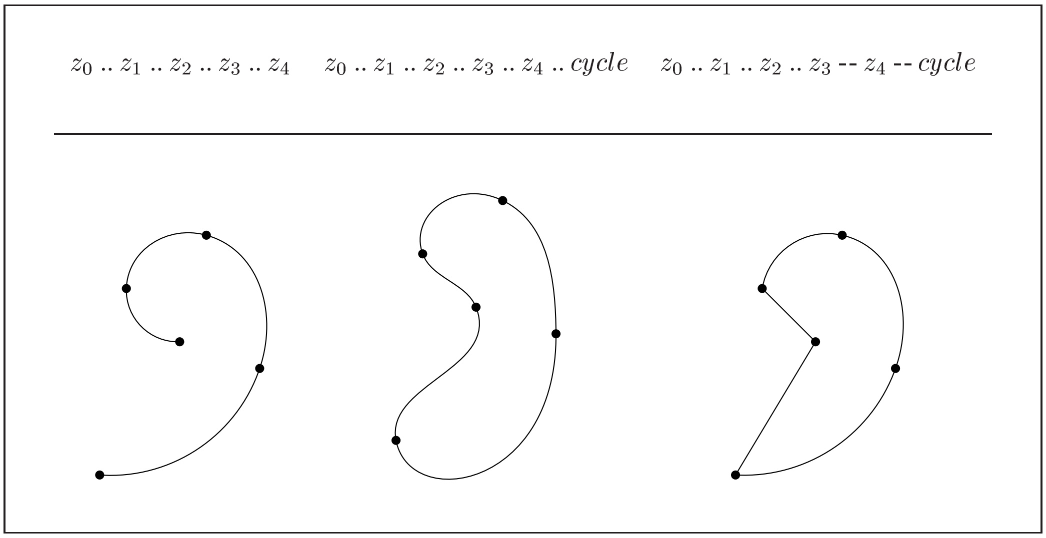
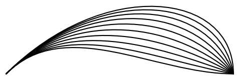


Figure 0.6.2: Different path connectors

$$
\begin{aligned}
\mathit{fig7} \quad &= [\mathit{vec}\ (0,0) \mathinner{..} \mathit{vec}\ (3 * \mathit{cm}, 0) \\
&\quad\ \#\ \mathit{setStartAngle}\ 45 \\
&\quad\ \#\ \mathit{setEndAngle}\ (-10 * a) \\
&\ \ |\, a \leftarrow [0\,..\,9]]
\end{aligned}
$$



$$
\begin{aligned}
\mathit{fig8} \quad &= [\mathit{vec}\ (0,0) \mathinner{..} \mathit{vec}\ (3 * \mathit{cm}, 0) \\
&\quad\ \#\ \mathit{setStartAngle}\ 45 \\
&\quad\ \#\ \mathit{setEndAngle}\ (10 * a) \\
&\ \ |\, a \leftarrow [0\,..\,7]]
\end{aligned}
$$

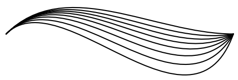


Figure 0.6.3: Start and end angle of path segments can be set

```
fig9 = rowSepBy 20 [(z0 .. z1
                       # setStartVector up
                       # setEndVector right) .. z2
                       # setEndVector down,
                    (z0 ... z1
                       # setStartVector up
                       # setEndVector right) ... z2
                       # setEndVector down]
```

Figure 0.6.4: The difference between the path connectors (..) and (...).

```
fig10 = rowSepBy 20 [
          z0 .. (z1 .. z2
                   # setJoin (joinTension (tension 1)))
          ..z3,
          z0 .. (z1 .. z2
                   # setJoin (joinTension (tension 2.3)))
          ..z3,
          z0 .. (z1 .. z2
                   # setJoin (joinTensions (tension 2.5)
                                           (tension 0.8)))
          ..z3]
```

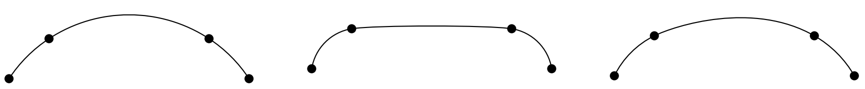


Figure 0.6.5: The tension "pulling" a path segments can be set

draw :: $[Path] \to Picture \to Picture$

0.7 Dash patterns

dashPattern :: $[Double] \to Pattern$

derived functions:

$$
\begin{array}{lll}
\mathit{dashed} & :: & \mathit{Pattern} \\
\mathit{dashed} & = & \mathit{dashPattern}\,[3,3]
\end{array}
$$

$$
\begin{array}{lll}
\mathit{dotted} & :: & \mathit{Pattern} \\
\mathit{dotted} & = & \mathit{dashPattern}\,[-1,2.5,0,2.5]
\end{array}
$$

0.8 Pencils

$$
\mathit{penSquare} \quad :: \quad (\mathit{Numeric},\mathit{Numeric}) \to \mathit{Numeric} \to \mathit{Pen}
$$

$$
\mathit{penCircle} \quad :: \quad (\mathit{Numeric},\mathit{Numeric}) \to \mathit{Numeric} \to \mathit{Pen}
$$

$$
\begin{array}{lll}
\mathit{calligraphic} & = & (\mathit{transformPath}\ (\mathit{scaled}\ 40)\ \mathit{fullcircle} \\
 & & \quad \#\mathit{setPen}\ (\mathit{penCircle}\ (0.1,5)\ 20)) \\
 & \square\square & (\mathit{transformPath}\ (\mathit{scaled}\ 40)\ \mathit{halfcircle} \\
 & & \quad \#\mathit{setPen}\ (\mathit{penSquare}\ (5,5)\ 45)) \\
 & \square\square & (\mathit{transformPath}\ (\mathit{scaled}\ 40)\ \mathit{quartercircle} \\
 & & \quad \#\mathit{setPen}\ 5)
\end{array}
$$

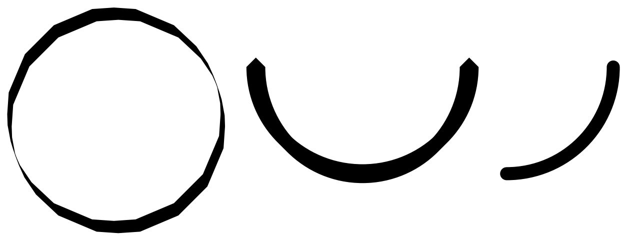


Figure 0.8.1: Calligraphic effects

$$
\begin{array}{llll}
\textit{hBrack} & & = & \textit{hBracket}\ (\textit{vec}\ (0,0))\ (\textit{vec}\ (100,0)) \\
& \textbf{where} & & \\
& \textit{hBracket pl pr} & = & \textit{pl} \\
& & & ...\textit{pl} + \textit{vec}\ (5,5) \\
& & & \text{---}\textit{med}\ 0.5\ \textit{pl pr} + \textit{vec}\ (-5,5) \\
& & & ...\textit{med}\ 0.5\ \textit{pl pr} + \textit{vec}\ (0,10) \\
& & & \&\textit{med}\ 0.5\ \textit{pl pr} + \textit{vec}\ (0,10) \\
& & & ...\textit{med}\ 0.5\ \textit{pl pr} + \textit{vec}\ (5,5) \\
& & & \text{---}\textit{pr} + \textit{vec}\ (-5,5) \\
& & & ...\textit{pr} \\
& & & \quad \#\ \textit{setPen}\ (\textit{penCircle}\ (0.001,1)\ 0)
\end{array}
$$

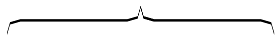


Figure 0.8.2: A horizontal bracket

0.9 Arrows

$$
\begin{array}{lll}
\textit{defaultArrowHead} & :: & \textit{PathArrowHead} \\
\textit{arrowHeadSize} & :: & \textit{Double} \to \textit{Double} \to \textit{PathArrowHead}
\end{array}
$$

derived functions:

$$
\begin{array}{lll}
\textit{arrowHeadBig} & :: & \textit{PathArrowHead} \\
\textit{arrowHeadBig} & = & \textit{pathArrowHeadSize}\ 8\ 4 \\
default & = & \textit{defaultArrowHead}
\end{array}
$$

different arrow styles

$$
\begin{array}{lll}
\textit{ahFilled}, \textit{ahLine} & :: & \textit{ArrowHeadStyle} \\
\textit{ahFilled} & = & \textit{AHFilled} \\
\textit{ahLine} & = & \textit{AHLine}
\end{array}
$$

$$
\begin{array}{lll}
\textit{setArrowHeadStyle} & :: & \textit{ArrowHeadStyle} \to \textit{ArrowHead} \to \textit{ArrowHead} \\
\textit{getArrowHeadStyle} & :: & \textit{ArrowHead} \to \textit{ArrowHeadStyle}
\end{array}
$$

```
bsp50                      = pointerChain 25 ["12", "2", "2", "103"]
    where
    pointerChain dx ps     = draw (backarrow : chainarrows)
                             (rowSepBy dx [b # setName (i :: Int)
                                          | (b, i) ← zip (map recBox ps) [0 ..]])
        where
        n                  = length ps
        backarrow          = arrow (ref (n − 1 ⋖ "bullet" ◃ C))
                                   (ref (n − 1 ⋖ "bullet" ◃ C) + vec (0, 20))
                                ---arrow (ref (0 ⋖ W) + vec (0, 20))
                                         (ref (0 ⋖ W))
        chainarrows        = [arrow (ref (i ⋖ "bullet" ◃ C))
                                    (ref (i + 1 ⋖ W))
                             | i ← [0 .. n − 2]]
        recBox a           =  (box a # setHeight (16 ∗ pt))
                           ◫  (box (toPicture bullet # setName "bullet")
                                 # setHeight (16 ∗ pt) # setWidth (16 ∗ pt))
```

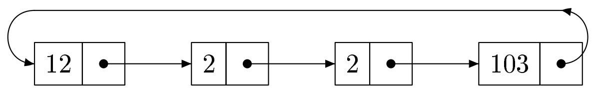


Figure 0.9.1: A linked list

0.10 Names

data *Dir* $= C \mid N \mid NE \mid E \mid SE \mid S \mid SW \mid W \mid NW$

Reference points: C,S,N,W,E.., ref

setName $::$ $\mathit{Name} \to \mathit{Picture} \to \mathit{Picture}$

$(\triangleleft)$ $::$ $\mathit{Name} \to \mathit{Name} \to \mathit{Name}$
$(\lessdot)$ $::$ $\mathit{Int} \to \mathit{Name} \to \mathit{Name}$

0.11 Areas

type Area

fill $::$ $[\mathit{Area}] \to \mathit{Picture} \to \mathit{Picture}$

```
bsp4 = let p = vec (−1 ∗ cm, 0)
               .. vec (0, −1 ∗ cm)
               .. vec (1 ∗ cm, 0)
               # setStartVector down
               # setEndVector up
       in
       fill [(p .. vec (0, 0)
                # setEndVector (vec (−1, −2)))
             .. cycle
               # setEndVector up]
       (p .. vec (0, 1 ∗ cm) .. cycle)
```

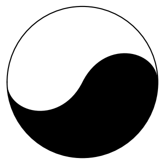


Figure 0.11.1: Arbitrary cyclic paths can be filled.

```
bsp5 = box (math "U"
         ▭ooalign
           [toPicture [toArea a # setColor 0.7,
                       toArea b # setColor 0.7,
                       toArea ab # setColor 0.4],
            aOverB])
  where
  aOverB = column [math "B" # setBGColor white,
                   vspace 50,
                   math "A" # setBGColor white]
  a      = transformPath (scaled 30) fullcircle
  aa     = transformPath (scaled 30) halfcircle
  b      = transformPath (scaled 30 & shifted (0, −30))
                         fullcircle
  ab     = buildCycle aa b
```

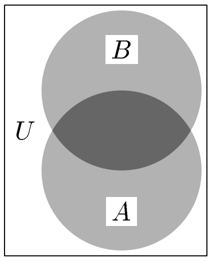


Figure 0.11.2: Another Example

0.12 Clipping

clip :: *Path* → *Picture* → *Picture*

0.13 Symbolic equations

define :: [*Equations*] → *Picture* → *Picture*
define :: [*Equations*] → *Path* → *Path*
define :: [*Equations*] → *Area* → *Area*

type Equation

$(\doteq)$ :: *a* → *a* → *Equation*
equal :: [*a*] → *Equation*

$(\doteqdot)$ :: *a* → *a* → *Boolean*
$(\dot{\not\equiv})$ :: *a* → *a* → *Boolean*
$(\dot{<})$ :: *a* → *a* → *Boolean*
$(\dot{\leqslant})$ :: *a* → *a* → *Boolean*

whatever :: *Point*

whatever :: *Numeric*

```
bsp3          =  define [
     equal [ref "z1", −ref "z2", vec (0.2 ∗ inch, 0)],
     equal [xpart (ref "z3"), −xpart (ref "z6"), 0.3 ∗ inch],
     equal [xpart (ref "z3") + ypart (ref "z3"),
            xpart (ref "z6") + ypart (ref "z6"), 1.1 ∗ inch],
     ref "z4"    ≐  med (1 / 3) (ref "z3") (ref "z6"),
     ref "z5"    ≐  med (2 / 3) (ref "z3") (ref "z6"),
     equal [ref "z20",
            med whatever (ref "z1") (ref "z3"),
            med whatever (ref "z2") (ref "z4")],
     equal [ref "z30",
            med whatever (ref "z1") (ref "z4"),
            med whatever (ref "z2") (ref "z5")],
     equal [ref "z40",
            med whatever (ref "z1") (ref "z5"),
            med whatever (ref "z2") (ref "z6")]]
                 (toPicture ["z1" -- "z20"
                         --"z2" -- "z30"
                         --"z1" -- "z40"
                         --"z2",
                           "z1" -- "z2"
                           # setPen 1,
                           "z3" -- "z6"
                           # setPen 1])
```

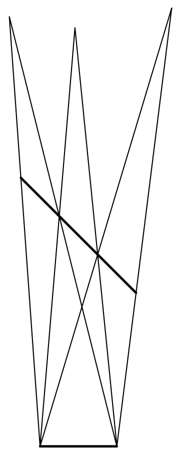
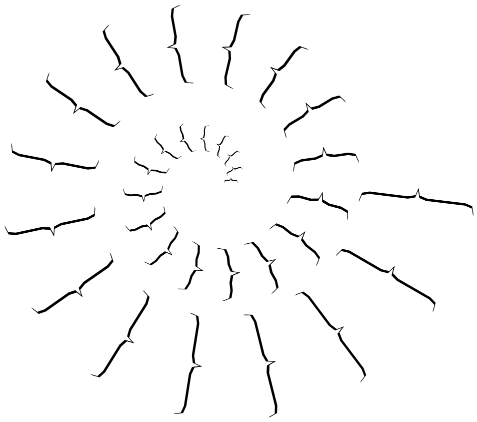


Figure 0.13.1: Unknown values are derived automatically by equation solving.

$$
\begin{array}{lll}
\mathit{brack} & = & [\mathit{bracket}\ ((5 + x\ /\ 14)\ \dot{*}\ \mathit{dir}\ x, (10 + x\ /\ 8)\ \dot{*}\ \mathit{dir}\ x) \\
 & & |x \leftarrow [0, 23\mathinner{.\,.}720]] \\
 & & \\
\mathit{bracket} & :: & (\mathit{Point}, \mathit{Point}) \rightarrow \mathit{Path} \\
\mathit{bracket}\ (\mathit{pl}, \mathit{pr}) & = & \mathit{define}\ [\\
\qquad \mathit{ref}\ \texttt{"start"} & \doteq & \mathit{pl}, \\
\qquad \mathit{ref}\ \texttt{"end"} & \doteq & \mathit{pr}, \\
\qquad \mathit{var}\ \texttt{"ang"} & \doteq & \mathit{angle}\ (\mathit{ref}\ \texttt{"start"} - \mathit{ref}\ \texttt{"end"}), \\
\qquad \mathit{var}\ \texttt{"d"} & \doteq & \mathit{cond}\ (\mathit{dist}\ (\mathit{ref}\ \texttt{"start"})\ (\mathit{ref}\ \texttt{"end"})\ \dot{<}\ 20) \\
 & & \qquad (\mathit{dist}\ (\mathit{ref}\ \texttt{"start"})\ (\mathit{ref}\ \texttt{"end"})\ /\ 4) \\
 & & \qquad 5, \\
\qquad \mathit{ref}\ \texttt{"vecl"} & \doteq & \mathit{var}\ \texttt{"d"}\ \dot{*}\ \mathit{dir}\ (\mathit{var}\ \texttt{"ang"} - 135), \\
\qquad \mathit{ref}\ \texttt{"vecr"} & \doteq & \mathit{var}\ \texttt{"d"}\ \dot{*}\ \mathit{dir}\ (\mathit{var}\ \texttt{"ang"} - 45), \\
\qquad \mathit{ref}\ \texttt{"start2"} & \doteq & \mathit{ref}\ \texttt{"start"} + \mathit{ref}\ \texttt{"vecl"}, \\
\qquad \mathit{ref}\ \texttt{"end2"} & \doteq & \mathit{ref}\ \texttt{"end"} + \mathit{ref}\ \texttt{"vecr"}, \\
\qquad \mathit{ref}\ \texttt{"mid"} & \doteq & \mathit{med}\ 0.5\ (\mathit{ref}\ \texttt{"start"})\ (\mathit{ref}\ \texttt{"end"}) \\
 & & \qquad +(1.41 * \mathit{var}\ \texttt{"d"}) \\
 & & \qquad \dot{*}\mathit{dir}\ (\mathit{var}\ \texttt{"ang"} - 90), \\
\qquad \mathit{ref}\ \texttt{"midl"} & \doteq & \mathit{ref}\ \texttt{"mid"} - \mathit{ref}\ \texttt{"vecl"}, \\
\qquad \mathit{ref}\ \texttt{"midr"} & \doteq & \mathit{ref}\ \texttt{"mid"} - \mathit{ref}\ \texttt{"vecr"}] \\
\end{array}
$$

$$
\begin{array}{l}
\qquad (\mathit{pl} \ldots \mathit{ref}\ \texttt{"start2"} \texttt{---} \mathit{ref}\ \texttt{"midl"} \ldots \mathit{ref}\ \texttt{"mid"} \\
\quad \&\mathit{ref}\ \texttt{"mid"} \ldots \mathit{ref}\ \texttt{"midr"} \texttt{---} \mathit{ref}\ \texttt{"end2"} \ldots \mathit{pr} \\
\qquad \#\ \mathit{setPen}\ (\mathit{penCircle}\ (0.001, \mathit{var}\ \texttt{"d"}\ /\ 5)\ (\mathit{var}\ \texttt{"ang"})))
\end{array}
$$

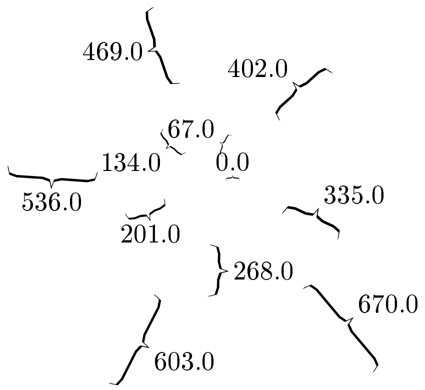


Example 0.13.1: Brackets

$$
\mathit{overlay} \quad :: \quad [\mathit{Equations}] \rightarrow [\mathit{Picture}] \rightarrow \mathit{Picture}
$$

```
brackLabel                   = [bracket′ x
                                    ((5 + x / 14) ∗̇ dir x, (10 + x / 8) ∗̇ dir x)
                                 |x ← [0, 67 .. 720]]

bracket′                     :: IsPicture a ⇒ a → (Point, Point) → Path
bracket′ l (pl, pr)          = bracket (pl, pr)
                                 #setLabel 0.5 C label
    where
    label                    = overlay′ [
            var "ang"        ≐ angle (pl − pr),
            ref (0 ◃ C)      ≐ cond (var "ang" <̇ (−175.5)
                                              +175.5 <̇ var "ang")
                                                   (ref (1 ◃ S))
                  (cond (var "ang" <̇ (−112.5)) (ref (1 ◃ SE))
                   (cond (   var "ang" <̇ (−67.5)) (ref (1 ◃ E))
                    (cond (var "ang" <̇ (−22.5)) (ref (1 ◃ NE))
                     (cond (var "ang" <̇ 22.5) (ref (1 ◃ N))
                      (cond (var "ang" <̇ 67.5) (ref (1 ◃ NW))
                       (cond (var "ang" <̇ 112.5) (ref (1 ◃ W))
                                 (ref (1 ◃ SW))
                             )   ))))))] (Just 0)
                                     [empty, toPicture l]
```



Example 0.13.2: Brackets with “intelligent” labels

0.14 Transformations

```
scale          :: Double → Picture → Picture
rotate         :: Double → Picture → Picture
```

$$
\begin{array}{lll}
\textit{fig28} & = & \textit{rek}\ 4\ \textit{empty} \\
\quad \textbf{where} & & \\
\quad \textit{rek}\ 0\ \textit{pic} & = & \textit{pic} \\
\quad \textit{rek}\ n\ \textit{pic} & = & \textit{ooalign}\ [\textit{draw}\ [p]\ [\textit{toArea}\ a\ \#\ \textit{setColor}\ 0.6, \\
 & & \qquad \textit{toArea}\ p\ \#\ \textit{setColor}\ \textit{white}], \\
 & & \qquad \textit{rotate}\ 90\ (\textit{scale}\ (1\ /\ 3)\ (\textit{rek}\ (n-1)\ \textit{pic}))] \\
\quad p & = & \textit{transformPath}\ (\textit{scaled}\ 30)\ \textit{fullcircle} \\
\quad a & = & (\textit{vec}\ (90,0)\ ..\ \textit{vec}\ (0,30)\ ..\ \textit{vec}\ (-90,0) \\
 & & \quad \#\ \textit{setEndCurl}\ 1) \\
 & & \quad ..\ \textit{vec}\ (0,-30)\ ..\ \textit{cycle} \\
 & & \qquad \#\ \textit{setEndCurl}\ 1
\end{array}
$$

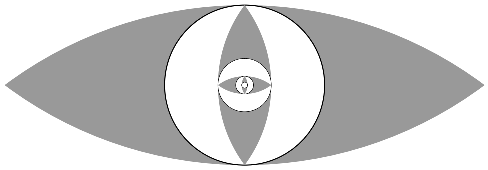
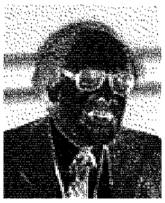
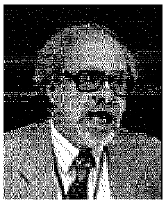


Figure 0.14.1: A recursive picture.

```
inverse [] = []
inverse (b : bs) = inv b : inverse bs
    where
    inv '0' = 'f'
    inv '1' = 'e'
    inv '2' = 'd'
    inv '3' = 'c'
    inv '4' = 'b'
    inv '5' = 'a'
    inv '6' = '9'
    inv '7' = '8'
    inv '8' = '7'
    inv '9' = '6'
    inv 'a' = '5'
    inv 'b' = '4'
    inv 'c' = '3'
    inv 'd' = '2'
    inv 'e' = '1'
    inv 'f' = '0'
    inv _ = '0'
figBit0 = image Depth1 figBit'
figBit1 = image Depth1 (map inverse figBit')
  -- The image is defined as an array of hex strings:
  -- figBit' = [
  -- "0000000000000000000000000000000000000000000000000000000000000000",
  -- "0000100000000000020000000000000000000000000000000000000000000000",
  -- ... 300 more such lines ... ]
figBitc = image Depth24 colorBit
  -- colorBit =[
  -- "edcfb1ffe2bbf7e2b1ffe2bbffedcfe2c59effe2bbffe2bbffedcfedcfb1f",
  -- "fa7ffe2bbf7d9b1edcfa7e2c59ee2c59eedcfb1a79e80bba7899e89769380",
  -- ... etc. ... ]
```



figbit0



figbit1

figBitc

Figure 0.14.2: Another recursive picture.